Testing and Evaluation

Assignment 1: Implementing and Comparing Sorting Algorithms

**Test1.txt**

Insertion sort comparison counter 1: 73

Input Array 1 Sort

007 012 018 025 041 051 057 065 066 073 097 101 103 110 118

Quicksort comparison counter 1: 83

Input Array 1 Sort

007 012 018 025 041 051 057 065 066 073 097 101 103 110 118

**Test2.txt**

Insertion sort comparison counter 2: 30

Input Array 2 Sort

005 014 018 025 041 043 059 065 066 073 097 103 110 112 118

Quicksort comparison counter 2: 108

Input Array 2 Sort

005 014 018 025 041 043 059 065 066 073 097 103 110 112 118

**Test3.txt**

Insertion sort comparison counter 3: 2840

Input Array 3 Sort

001 001 001 001 002 003 005 006 009 011 020 020 020 020 021 023 028 029 030 033

033 043 045 046 046 046 046 046 055 055 055 056 057 061 063 066 069 069 070 071

072 073 076 077 079 080 081 083 085 088 091 092 094 094 095 099 101 101 103 105

106 107 110 113 118 125 127 128 129 136 138 140 143 144 147 148 150 150 153 156

158 169 169 169 170 171 175 178 180 184 184 184 184 189 190 193 198 199 199 199

Quicksort comparison counter 3: 1008

Input Array 3 Sort

001 001 001 001 002 003 005 006 009 011 020 020 020 020 021 023 028 029 030 033

033 043 045 046 046 046 046 046 055 055 055 056 057 061 063 066 069 069 070 071

072 073 076 077 079 080 081 083 085 088 091 092 094 094 095 099 101 101 103 105

106 107 110 113 118 125 127 128 129 136 138 140 143 144 147 148 150 150 153 156

158 169 169 169 170 171 175 178 180 184 184 184 184 189 190 193 198 199 199 199

**Test4.txt**

Insertion sort comparison counter 4: 363

Input Array 4 Sort

001 001 001 003 003 003 003 005 005 006 009 009 009 011 021 021 021 028 028 029

030 033 033 033 041 041 041 041 041 043 043 054 055 055 056 056 056 057 060 060

063 066 067 069 069 070 071 073 074 074 079 080 080 080 080 085 091 091 094 094

094 094 094 095 099 101 101 101 101 101 103 105 107 115 115 115 115 118 127 127

136 136 138 147 148 148 148 150 152 152 152 163 169 169 170 170 170 180 190 190

Quicksort comparison counter 4: 1563

Input Array 4 Sort

001 001 001 003 003 003 003 005 005 006 009 009 009 011 021 021 021 028 028 029

030 033 033 033 041 041 041 041 041 043 043 054 055 055 056 056 056 057 060 060

063 066 067 069 069 070 071 073 074 074 079 080 080 080 080 085 091 091 094 094

094 094 094 095 099 101 101 101 101 101 103 105 107 115 115 115 115 118 127 127

136 136 138 147 148 148 148 150 152 152 152 163 169 169 170 170 170 180 190 190

Task 3 – Five Statements

1. Insertion sort performs better than quicksort for smaller test samples as shown in the number of comparisons in Test1.txt and Test2.txt.
2. Insertion sort performs better in Test4.txt this may be because this data sample is partially sorted before being sorted.
3. Quicksort performed better than insertion sort in Test3.txt, I think that this is because the pivot value is a central value and also the values are more randomised in the data sample before being sorted.
4. I think that quicksort performs worse when the pivot is the largest number in the data sample, this can be shown in Test3.txt.
5. Quicksort performs worse on Test4.txt, this might be due to the fact that this data sample is already partially sorted before being sorted.

Further Testing

**Test3.txt**

New sort comparison counter 3: 7388

Input Array 3 Sort

001 001 001 001 002 003 005 006 009 011 020 020 020 020 021 023 028 029 030 033

033 043 045 046 046 046 046 046 055 055 055 056 057 061 063 066 069 069 070 071

072 073 076 077 079 080 081 083 085 088 091 092 094 094 095 099 101 101 103 105

106 107 110 113 118 125 127 128 129 136 138 140 143 144 147 148 150 150 153 156

158 169 169 169 170 171 175 178 180 184 184 184 184 189 190 193 198 199 199 199

**Test4.txt**

New sort comparison counter 4: 4971

Input Array 4 Sort

001 001 001 003 003 003 003 005 005 006 009 009 009 011 021 021 021 028 028 029

030 033 033 033 041 041 041 041 041 043 043 054 055 055 056 056 056 057 060 060

063 066 067 069 069 070 071 073 074 074 079 080 080 080 080 085 091 091 094 094

094 094 094 095 099 101 101 101 101 101 103 105 107 115 115 115 115 118 127 127

136 136 138 147 148 148 148 150 152 152 152 163 169 169 170 170 170 180 190 190

**Test5.txt**

New sort comparison counter 5: 1682

Input Array 5 Sort

001 001 001 001 001 001 001 001 020 020 020 020 028 028 028 028 028 028 028 046

046 046 046 046 046 046 046 055 055 055 055 055 069 069 069 069 069 069 072 072

072 072 079 079 079 079 079 099 099 099 099 099 099 099 099 099 107 107 107 107

107 107 107 127 127 127 127 127 127 150 150 150 150 150 150 150 150 153 153 153

153 169 169 169 169 169 184 184 184 184 184 184 184 184 199 199 199 199 199 199

Insertion sort comparison counter 5: 2728

Input Array 5 Sort

001 001 001 001 001 001 001 001 020 020 020 020 028 028 028 028 028 028 028 046

046 046 046 046 046 046 046 055 055 055 055 055 069 069 069 069 069 069 072 072

072 072 079 079 079 079 079 099 099 099 099 099 099 099 099 099 107 107 107 107

107 107 107 127 127 127 127 127 127 150 150 150 150 150 150 150 150 153 153 153

153 169 169 169 169 169 184 184 184 184 184 184 184 184 199 199 199 199 199 199

153 169 169 169 169 169 184 184 184 184 184 184 184 184 199 199 199 199 199 199

5a) New sort algorithm performs better in Test class 5. I think that this is because of the amount of duplicate numbers in the array.

5b) The worst case for big Oh in the new sort algorithm is O(N­2), this happens when there are no duplicates.

5c) The best case for big Oh in the new sort algorithm is O(N), this happens when all values in the array are duplicates of each other.